1 Boolean Algebra

Student Group

First Name	Surname	Matrikel Nr.

Table of Contents

1. Boolean Algebra

1.0 Motivation: Digital electronics in daily life

Goals

After this lesson you should:

1. know the Boolean functions, their notations, and truth tables.
2. be able to apply the Boolean rules of arithmetic.
3. be able to simplify Boolean expressions.
4. know the following terms: (logic) gates, names of arithmetic rules.

1.0.1 Everything is one exept zero...

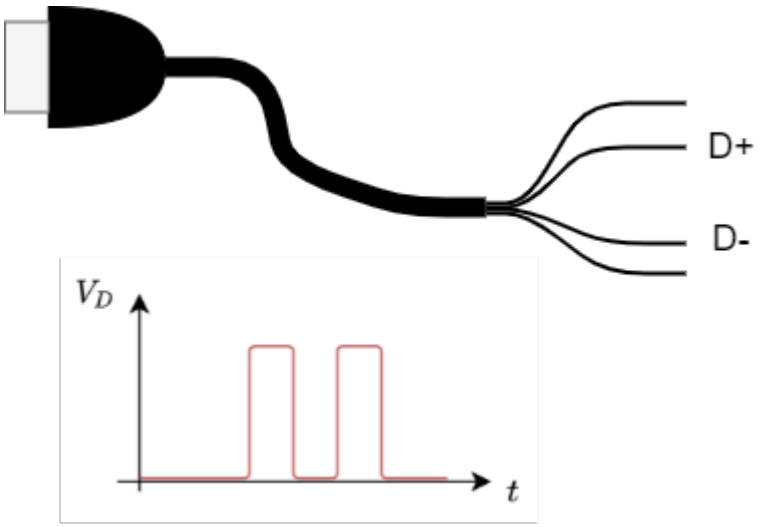


Fig. 1: first Example: USB cable

Before we start to dive deeper into the digital systems, it is a good idea to approach the topic with a first practical example. For this, we have a look onto an USB cable and mentally cut the cable. We will see a total of four wires waiting for us. Two of them are called D+ and D-. These are the wires that are used for digital data transmission. With special tools - like an oscilloscope we can measure the time course of the voltage between D+ and D-. This voltage is shows two different levels and rises and falls in between. What we see are the **logic states** 0 or 1!

Beyond data transmission, we can also encode other things with binary numbers: for example, we could look at

- the position of a lever - whether it is pointing up or down.
- or a switch that is open or closed.
- or a light that is on or off.
- even whether it is raining or not raining, whether a dog is barking or not barking or other things - we can encode all of that in binary units.

In reality, the measured voltage on the USB cable would also show noise and only roughly depict the two states. When we ignore the noise and only assume that an upper level (HIGH or H) encode one state and the lower one (LOW or L) the second state the diagram ist also called the **level diagram**. In contrast to analog systems, a signal can only take one of the two valid states in digital systems. Similarly, boolean algebra only uses the states TRUE and FALSE.

Note!

- The smallest binary message set is called a “**bit**”, which comes from the English term “binary digit”. A bit thus describes the logical state of a two-valued system. The binary characters can be written as, e.g.:
  - '0', '1'
  - FALSE, TRUE
  - HIGH, LOW
- In binary logic, every expression can only be **true or false**. If something is not true, it has to be false - and vice versa.
- Therefore, binary logic has some **limitations for real world applications**: It may only rain softly, a light might be dimmed and also the voltage (which represents the bit) might be in an in-between state. This situation has to be coped with beyond the binary logic. Beyond binary logic, there is often an **invalid state** in reality, which separates the logic states.

We will find out more of the details behind this states in the chapter Number Systems. In this chapter we will start to think about, how this binary signals in connection with some algebra can generate a base for fundamental logic building blocks. This building blocks will be stacked together to larger boxes in the next chapters and are the basis for microcontroller, microprocessors and virtually all digital electronics from watches over automotive controllers to super computers.

Fig. 2: Inner 'Life' of an Integrated Cicuit

1.0.2 At the heart of a computer

Fig. 3: A 'topless' microcontroller

The core of a computer is the processor in which the instructions are executed. This central processing unit (CPU) is also used in microcontrollers, which can be found around us in almost every device: Mobile phones, cars, bank cards, washing machines... Often there are even several microcontrollers installed in the devices.

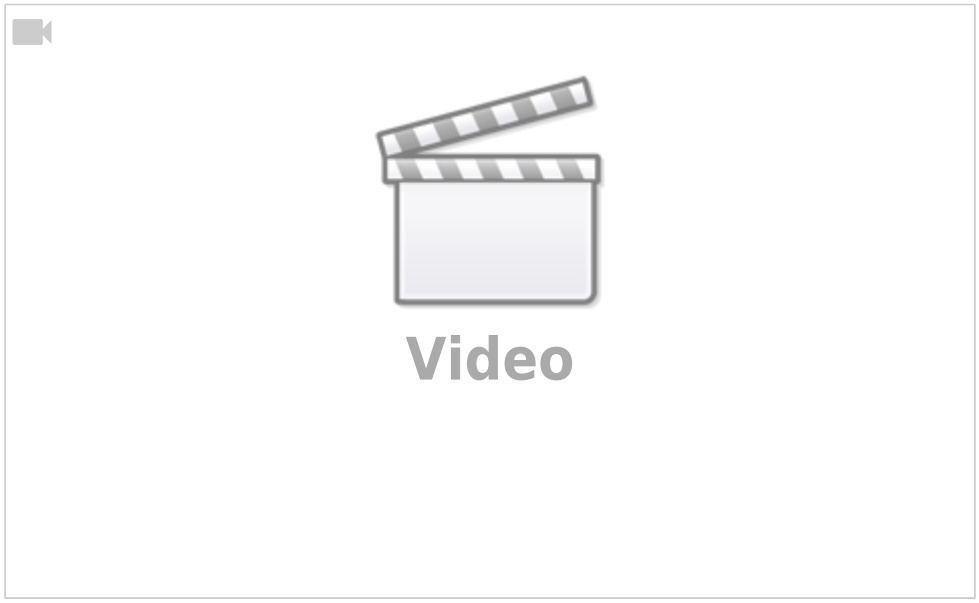
In the microcontroller, in addition to the command-executing microprocessor (more precisely, the {wp>arithmetic-logic unit}), other peripherals such as memory, clock generation, analog-to-digital converter and much more are built in. This makes it a compact tool for many applications. If you look at the microcontroller under an optical microscope, you will see the following picture (figure 3).

Fig. 4: Microcontroller under the microscope

Fig. 5: Microcontroller schematic

But now let's take a look at the structure of the processor. The processor shown in figure 4 uad figure 5 was developed by two students in 1990 and consists of several tens of thousands of transistors. This chip paved the way to cheap, fast and yet easily programmable controllers, from the fax machine to the hobby basement, and can be found on the Arduino boards, among others. You will get to know and to program the ATmega328 - a distant successor with several hundred thousand transistors - in higher semesters. The images depict various peripheral components: The processor, FLASH, EEPROM and fuses will be shown in the following chapters. The voltage supply (German 'Spannungsversorung'), is needed to generate additional higher voltages based on the supplied external voltage.

The following clip shows a zoom into the smallest parts of the controller.



One question is still unclear: how can we connect the zeros and ones in such a way, that the processor can calculate something like $23 + 42$? For this we need to have a look onto binary logic.

1.2 Binary Logic

Goals

After this lesson you should:

1. know the Boolean functions, their notations, and truth tables.
2. be able to apply the Boolean rules of arithmetic.
3. be able to simplify Boolean expressions.
4. know the following terms: (logic) gates, names of arithmetic rules.

1.2.1 First steps into logic

We have already learned about the 'bit', and it's two-valued value. This can be connected to the ancient idea of a binary logic. The Greek Philosopher Aristotle started to build up a systematic in order to conclude from statements like 'at night, it is dark outside' and 'it is night' to 'it has to be dark outside'. It might seem a bit unrelated to comtrollers and computers, at the first sight. But in this scientific interpretation of logic all logic statements are either true or false.

George Boole developed a more mathematical way into handline logic. Based on his work the fundamental logic was solidified into axioms. One axiom we have already seen: If something is true, it cannot be false ans vice versa (the 'theorem of contradiction').

Other axioms help to combine statements. This is called 'reasoning' or 'deduction'.

- This was already used some sentences before: If 'at night, it is dark outside' and 'it is night' is true, one can make the **implication** 'it has to be dark outside'.
  But be aware, that is is not always reversible: It could also be a solar eclipse.
- Another deduction is the **equivalence**: when 'A' implies 'B', and 'B' implies 'A' both statements are equivalent.

In the following, we will have a look onto the basic logic combinations, which are needed for digital systems. These basic logic combinations of logic statements (or bits) are called **logic operators** and can be used similarly to the algebraic “plus” or “minus”. The mathematical construct, which describes the systematic of these combinations is called **boolean algebra**. In digital systems the logic operators can be represented as a circuit of transistors or switches. This can be blackboxed into **logic gates**.

This tools will get more familiar in the next subcapter.

1.2.2 The logic operator NOT

Fig. 6: Simulation of an Inverter

The first very simple circuit negates the input value. It is also called an inverter or negation (logic NOT, NOT gate). This logic operator always generates the output value $Y(0)=1$ from the digital input value $X=0$ and for $X=1$ correspondingly $Y(1)=0$. figure 6 shows an example: The light is only on ($X=1$), when the input is off ($Y=0$) - this functionality is “hard-wired”.

When you think about the inputs and outputs in the shown picture above, you realise, that the input is a voltage, but the output is a current. This is sometimes beneficial, but inside an digital systems commonly only voltages are used.

Fig. 7: Simulation of an Inverter

In order to control a (output) voltage with a (input) voltage two complement types of switches are combined similar to a voltage divider or a half bridge (see figure 7). One is normally open and the other one normally closed. This can also set up with complement types of transistors. Thus, only one transistor (TRANSfer ResISTOR) becomes conductive at a time, the other one correspondingly high impedance.

With this setup the logic voltages ($0V$, $5V$) are just switched complementary via the switches. This technique is also called **CMOS** technique: Complementary MOSFET. In today's electronics, this technology is used throughout and has completely replaced older variants (e.g. TTL).

Comparing the circuits in figure 6 and figure 7, one could simply see, that double the amount of switches have to be used and the lower one is a bit trickier to understand. With this in mind, for the following operators only the first type of circuit will be shown.

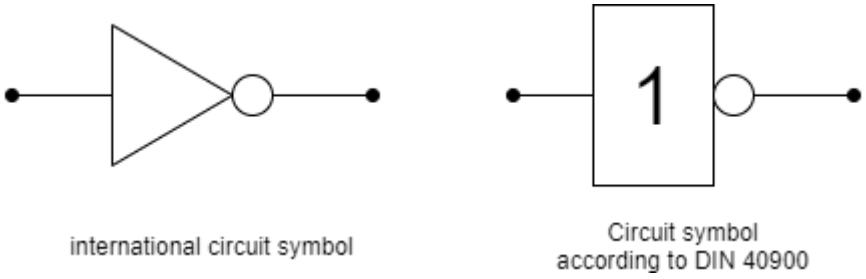


Fig. 8: Negation: Circuit symbols

The circuit symbols of the NOT-gate are shown in figure 8. Depending on the (software) tools and schematics one of the type is printed.

NOT	
X	Y
0	1
1	0

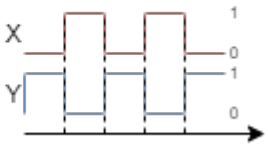


Fig. 9: Negation: Truth table and timing diagram

Besides the symbol other representation are also common (see also figure 9):

- The **truth table** shows the input(s) X on the left and the output(s) Y on the right. There is a row for each distinct combination of inputs. This representation will get handy in the next chapters, in order to analyse more complex logic.
- The **timing diagram** shows the sequential behavior. For this diagram, the input variables are stimulated with all possible state combinations. Also this will become handy especialy in the chapter Sequential Logic.
- In **math** the inversion has also multiple representations e.g.
  - $Y = /X$ (used e.g. for input with a keyboard)
  - $Y = \overline{X}$ (often used when handwriting and math)
  - Y = !X (used in c language)

Note!

- Inputs are always denoted with X
- Outputs are always denoted with Y
- There are multiple ways for representing the logic. The most common ones are:
  electric ciruit with switches (or transistors), logic gate, truth table, timing diagram, mathematical representation.
- When you use a representation: use it uniformly.

1.2.3 The logic operator AND

Fig. 10: Simulation of a conjunction

The next circuit will generate an positive output only, if all inputs are true. This is called a logical **conjunction** (logic AND, AND gate). When one ore more inputs are false the output is also false. figure 10 shows an example: The light is only on ($Y=1$), when all inputs are on ($X0=1$, $X1=1$, ...). This is commonly used for safety circuits, e.g. when the workspace of a robot has multiple doors and all have to be closed in order to start.

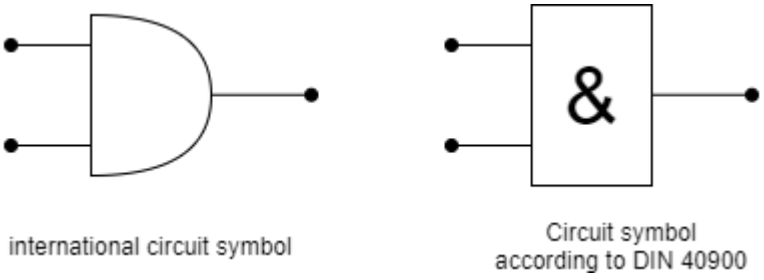


Fig. 11: Conjunction: Circuit symbols

AND		
X1	X0	Y
0	0	0
0	1	0
1	0	0
1	1	1

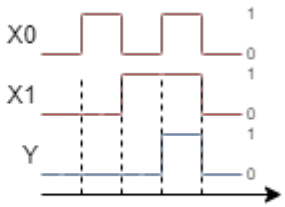


Fig. 12: Conjunction: Truth table and timing diagram

Again, the other representation are shown:

- The truth table and timing diagram are depicted in figure 12.
- In **math** the AND operation has again multiple representations e.g.
  - $Y = X0 * X1$ (used e.g. for input with a keyboard)
  - $Y = X_0 \cdot X_1$ (often used when handwriting or in math)
  - `Y = X0 && X1` (used in c language), `Y = X0 & X1` (used in some other languages)

- $Y = X_0 \land X_1$ (used in logic)

For upcoming, more complex terms the algebraic notation ($Y = X_0 \cdot X_1$) usually lead to a better understanding.

1.2.4 The logic operator NAND

Fig. 13: Simulation of a NAND operation

The NOT gate is often used in front of or after other gates. When used after AND gates, this creates an 'NOT AND' or in short 'NAND' (logic NAND, NAND gate). This circuit will only generate an negative output only, if all inputs are true. When one ore more inputs are false the output is true. figure 13 shows an example: The light is only off ($Y=0$), when all inputs are high ($X0=1$, $X1=1$, ...). In the simulation one has to look in detail: the used switches are normally closed (closed when the input is low). Therefore the switches are only open, when the input is high.

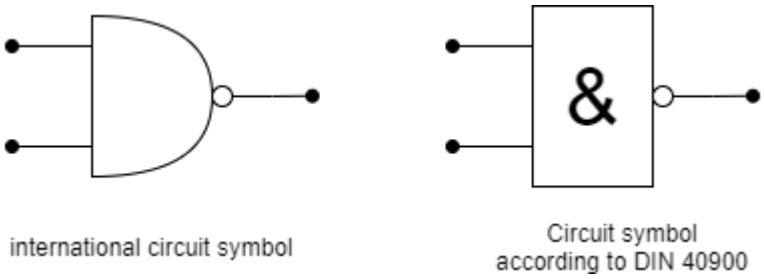


Fig. 14: NAND Circuit symbols

The circuit symbols are shown in figure 14. In order to shorten the ciruit, the NOT is often 'shrinked' only to a small circle after the gate.

NAND		
X1	**X0**	**Y**
0	0	1
0	1	1
1	0	1
1	1	0

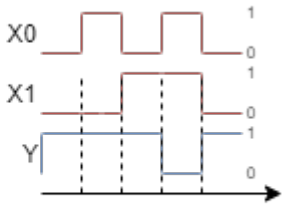


Fig. 15: NAND: truth table and timing diagram

Again, the other representation are shown:

- The truth table and timing diagram are depicted in figure 15.
- In **math** the NAND operation has again multiple representations e.g.
  - $Y = /(X0 * X1)$ (used e.g. for input with a keyboard)
  - $Y = /(X_0 \cdot X_1)$ or $Y = \overline{X_0 \cdot X_1}$ (often used when handwriting or

in math)
- `Y = !(X0 && X1)` (used in c language)
- $Y = /(X_0 \land X_1)$ or $Y = \overline{X_0 \land X_1}$ (used in logic)

1.2.5 The logic operator OR

Fig. 16: Simulation of a OR operation

We already had a look onto the AND gate. So what about OR? Of course there is also this kind of operation. This is called a logical **disjunction** (logic OR, OR gate). When there is one or more input true the output is also true. figure 16 shows an example: The light is only off ($Y=0$), when all inputs are on ($X0=1$, $X1=1$, ...). Doesn't it - at the first glimpe - seem similar to the NAND circuit? The main difference is that normally open switches are used. Only, when both switches are open the light is off.

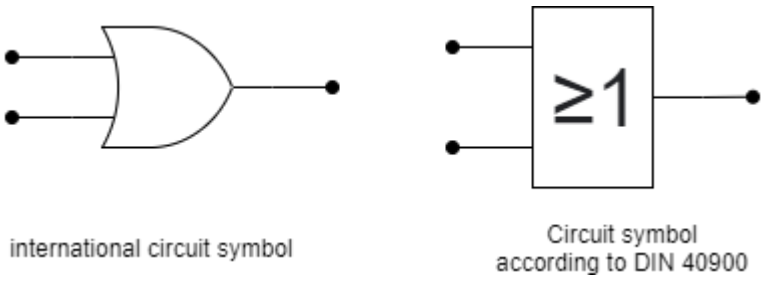


Fig. 17: OR circuit symbols

The circuit symbols are shown in figure 17. The DIN symbol is derived from the fact, that one ore more inputs have to be true to get an true output.

OR		
X1	**X0**	**Y**
0	0	0
0	**1**	**1**
1	0	**1**
1	**1**	**1**

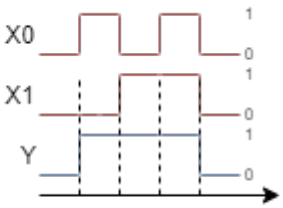


Fig. 18: OR: truth table and timing diagram

Again, the other representation are shown:

- The truth table and timing diagram are depicted in figure 18.
- In **math** the OR operation has again multiple representations e.g.
  - $Y = X0 + X1$ (used e.g. for input with a keyboard or in handwriting)
  - `Y = X0 || X1` (used in c language)
  - $Y = X_0 \lor X_1$ (used in logic, the $\lor$ stands for the Latin *vel*, which means or)

For upcoming, more complex terms the algebraic notation ($Y = X_0 + X_1$) usually lead to a better understanding. Also here: we will see the connection to math in the next chapters.

1.2.6 The logic operator XOR

Fig. 19: Simulation of a XOR operation

Beside the OR there is also an “either … or …, but not both”. This is called exclusive or, in short XOR (logic XOR, XOR gate). Only when one input is true the output is true. figure 16 shows an example: When none or when all inputs are on, the light is only off. The circuit looks a bit more complicated with two series branches in parallel and the use of both normally closed and normally open switches. On the other hand one can already think about, that one of the branches look similar to the setup for the AND gate, and the parallel setup similar to the NAND gate. Could it be possible to convert the gates into each other? We will see that next…

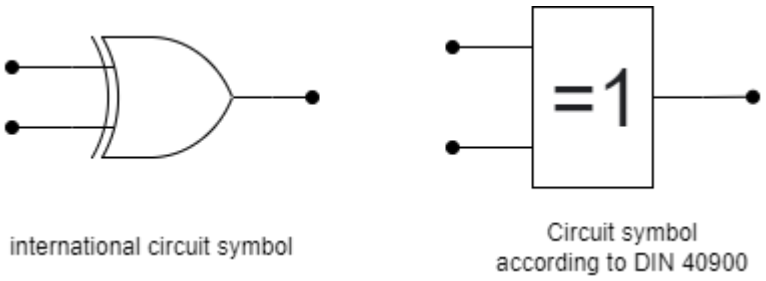


Fig. 20: OR circuit symbols

The circuit symbols are shown in figure 20. Both symbols have similarities with the OR symbols.

XOR		
X1	**X0**	**Y**
0	0	0
0	1	1
1	0	1
1	1	0

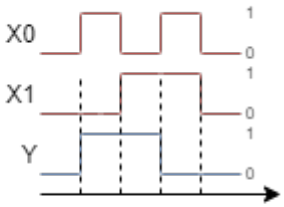


Fig. 21: OR: truth table and timing diagram

Again, the other representation are shown:

- The truth table and timing diagram are depicted in figure 21.
- In **math** the XOR operation has again multiple representations e.g.
  - $Y = X0 \# X1$ (used e.g. for input with a keyboard or in handwriting)
  - there is no bit operation in c language for XOR
  - $Y = X_0 \oplus X_1$ (used in logic)

Excercise 1.2.1. NOR and XNOR

1. Think about a circuit (with multiple switches and one lamp) to implement NOR and XNOR.
2. What is the relation between the circuits of NOR and AND? And how about XOR and XNOR?
3. What would the gate representation and the other representations look like?

1.3 Convertibility of gates

> **Goals**
>
> After this lesson you should:

Some of the circuits in the provious chapter looked suspiciously similar. We will now have a more deeper look onto this and try to converte some gates into each other. In this subchapter we will focus on combining NAND gates in order to build other gates. As we will see, based on the NAND and NOR gates any other gate and any other logic can be created.

1.3.1 NAND in NOT

Fig. 29: From NAND to NOT

The conversion from NAND to NOT is relatively simple: When both inputs to NAND are the same (either '1' or '0') the output will be the negation of the input. This can also be seen in the truth table of the NAND gate: when the inputs as the same only the first and last row, have to be considered and lead to a inverting behaviour.

A different approach to get an NOT is to set the second input to '1'. Here, the NOT can be 'deactivated' of with the second input. This can be tested in the figure 29 by clicking on the input 'H' of the NAND gate on the right below.

1.3.2 NAND in AND

Fig. 30: From NAND to AND

With the knwoledge from 'NAND to NOT' the NAND can be converted to AND: a negated NAND leads to an AND. It is roughly similar to 'not a no-go' is logically a 'go'. Therefore, the NOT hat to be set

behind the NAND.

1.3.2 NAND in OR

Fig. 31: From NAND to OR

When each input of a NAND gate is inverted the result acts like an OR gate. In order to understand this, one can again look onto the truth table of the NAND and the OR gate and try to invetigate what happens when the inputs of the NAND are negated.

Excercise 1.3.1. further conversions

1.4 Rules for boolean algebra

Goals

After this lesson you should:

1. Understand the purpose of the Tri-State gate and the “Z” state.
2. Understand the use of the “Don't care” state.
3. Be able to convert interconnections from a few logic gates to truth tables and vice versa.
4. Be able to trace gates to NAND and NOR gates.

We have seen, that (at least) some of the gates can be represented by means of others. In order to approach this more systematically, we will now have a look onto the arithmetic rules of boolean algebra. These rules can be used to either build a logic cicruit out of the basis gates shown in chapter 1.2. On the other hand we are also able to simplify the logic circuits by ths rules.

1.4.1 The set of rules

Nr	Math Term / Formula	Description
1	Closure $B \land B \rightarrow B $ $B \lor B \rightarrow B $	The operators $\land$ and $\lor$ map elements from $B=\{0,1\}$ to B.
2	Duality	If A is a statement of boolean algebra, so is A^* . A^* is obtained by exchanging $\land$ with $\lor$ and vice versa.

Nr	Math Term / Formula	Description
3	Neutral Element	There exist a neutral element to the operators $\land$ and $\lor$. Applying the oparator to a and the neutral element results in a.
	$a \land 1 = a $	
	$a \lor 1 = a $	
4	Complementary Element	There exist a complementary element to the operators $\land and $\lor$. The negation of a is for both operators the complementary element.
	$a \land \overline{a = 0} $	
	$a \lor \overline{a = 1} $	
5	Idempotence	Applying the operators $\land$ and $\lor$ to a similar input a results in a.
	$a \land a = a $	
	$a \lor a = a $	
6	Commutative Law	Inputs a and b are interchangable.
	$a \land b = b \land a $	
	$a \lor b = b \lor a $	
7	Associative Law	For the same operator bracketing can be moved.
	$a \land (b \land c) = (a \land b) \land c $	
	$a \lor (b \lor c) = (a \lor b) \lor c $	
8	Distributive Law	There exist a nuetral element to the operators \land and \lor.
	$a \land (b \lor c) = (a \land b) \lor (a \land c)$	
	$a \lor (b \land c) = (a \lor b) \land (a \lor c)$	
9	Law of Absorbtion	There exist a nuetral element to the operators \land and \lor.
	$a \land (a \lor b) = a$ $a \land (a \lor b) = a \land b $	
	$a \lor (a \land b) = a$ $a \lor (a \land b) = a \lor b $	
10	DeMorgan's Rule	there exist a nuetral element to the operators \land and \lor.
	$a \land 1 = a , a \lor 0 = a $	
	$a \land 1 = a , a \lor 0 = a $	

Nr	Math Term	Formula	Description
1	Closure:	$B \land B \rightarrow B $	The operators $\land$ and $\lor$ map elements from $B=\{0,1\}$ to B.
		$B \lor B \rightarrow B $	
2	Duality:		If A is a statement of boolean algebra, so is A^* . A^* is obtained by exchanging $\land$ with $\lor$ and vice versa.
3	Neutral Element	$a \land 1 = a $	There exist a neutral element to the operators $\land$ and $\lor$. Applying the oparator to a and the neutral element results in a.
		$a \lor 1 = a $	
4	Complementary Element	$a \land \overline{a = 0} $	There exist a complementary element to the operators $\land and $\lor$. The negation of a is for both operators the complementary element.
		$a \lor \overline{a = 1} $	
5	Idempotence	$a \land a = a $	Applying the operators $\land$ and $\lor$ to a similar input a results in a.
		$a \lor a = a $	
6	Commutative Law	$a \land b = b \land a $	Inputs a and b are interchangable.
		$a \lor b = b \lor a $	

Nr	Math Term	Formula	Description
7	Associative Law	$a \land (b \land c) = (a \land b) \land c $	For the same operator bracketing can be moved.
		$a \lor (b \lor c) = (a \lor b) \lor c $	
8	Distributive Law	$a \land (b \lor c) = (a \land b) \lor (a \land c)$	There exist a nuetral element to the operators \land and \lor.
		$a \lor (b \land c) = (a \lor b) \land (a \lor c)$	
9	Law of Absorbtion	$a \land (a \lor b) = a$ $a \land (a \lor b) = a \land b $	There exist a nuetral element to the operators \land and \lor.
		$a \lor (a \land b) = a$ $a \lor (a \land b) = a \lor b $	
10	DeMorgan's Rule	$a \land 1 = a , a \lor 0 = a $	there exist a nuetral element to the operators \land and \lor.

zwei Transistor(typ)en reichen

Fig. ##: n- und p-Kanal MOSFETs

Ok, now we know at least that the chip consists of many transistors. But how do they work and how can you build something as complex as a processor from them? The exact function is the content of the course '(Analog) Circuit Design'. For digital applications it is sufficient to have a simple picture of a certain type of transistor - the MOSFET - in your head. This has the three terminals:

- **S**ource (“Quelle”), the inflow of charge carriers
- **D**rain (“Senke”), the outflow of the charge carriers
- **G**ate (“Tor”), the gatekeeper, which regulates the passage between source and drain: If the correct voltage gate is present at the gate terminal [1)], the Source and Drain terminals are short-circuited, that is, a current can flow and the voltage drop between them becomes small.

Two types of MOSFET are important in the following digital circuits:

- one that is non-conductive for low voltages ($0V$, logic 0, Low, L, or False) at the gate (n-channel MOSFET) and
- one that is non-conductive for high voltages ($5V$, logic 1, High, H or True) at the gate (p-channel MOSFET).

In the picture on the right (figure ##) you can see the two variants in action, when the voltage at the gate just assumes the digital voltage values.

Now it is interesting, that these two kinds of transistors are sufficient to build all variants of logical functions. Logical functions combine one or more inputs ($X_1, ... X_n$) in such a way, that every kind of input uniquely leads to an output ($Y_1, ... Y_n$). In circuits, so-called gates correspond to logical

functions. They need not be only such simple functions like an AND-gate. Mathematical operations can also be mapped in this way. To do this, gates must be cleverly combined with each other. In an exercise on binary logic, it is shown that all gates can be constructed using NAND or NOR gates. So, if we could figure out how to build a NAND or NOR gate from transistors, we could in turn build all gates up to addition from it.

Gatter logisch?

Fig. 6: Simulation eines Inverters

Before the NAND gate is considered, a very simple circuit is to build, which generates the output value $Y(0)=1$ from the digital input value $X=0$ and for $X=1$ correspondingly $Y(1)=0$. This circuit always negates the input value and is called inverter. For this purpose the two types of transistors are combined similar to a voltage divider or a half bridge. Thus, only one transistor (TRANSfer ResISTOR) becomes conductive at a time, the other one correspondingly high impedance. In figure 6 the corresponding circuit with normally open and normally closed switch is also drawn. It is important for the following consideration that the logic voltages ($0V$, $5V$) are just switched complementary via the switches. For this reason this technique is also called CMOS technique: Complementary MOSFET. In today's electronics, this technology is used throughout and has completely replaced older variants (e.g. TTL).

Fig. 2: Inverter gate in CMOS on chip

From this example, it can be seen how a logical 1 can become 0 in mobile phones, vehicle control units, and television sets. The figure 2 shows the realization of this gate in silicon:

- Figure (1) shows the image of a scanning electron microscope, which shows several layers at the same time. The three circular elements are electrical passages (“vias”) through several layers. In green are the two structures of the MOSFETs, which can act as a “valve” to open the connection up ($5V$) or down ($0V$).
- Figure (2) is a false color image. In beige is the top conductive layer and in blue is the non-conductive region of the top layer.
- In figure (3), the different signals have been highlighted.
  The output A is taken out via a via with the right connection.

not-AND, or?

Fig. 7: Simulation of a CMOS NAND gate

But how does the NAND gate work, on which logic can be built it? The concept behind this gate is that the output will only output logic false ($Y=0$) if both inputs are set to logic true ($A=1$ and $B=1$). The gate is shown in figure 7 above. This must be implemented using the two types of transistors , which were already explained. So, this structure must be built in such a way that:

1. only if both inputs switch transistors with $A=1$ and $B=1$ at the same time, $0V$ shall be applied,
2. if one of the inputs A or B, or both equal 0, $5V$ shall be applied.

The first one is possible via a series connection of transistors at $0V$. These must short the source and drain terminals at a gate voltage of $5V$ (input to logic 1), so n-channel MOSFETs are required.
The second one requires a parallel connection of transistors against $5V$. These must short the source and drain with a gate voltage of $0V$. Here p-channel MOSFETs are used (figure 7 below).

The implementation in silicon (figure ##) again appears somewhat unclear at first glance. In this figure three pictures are to be seen, again. In the first picture the transistors are marked green again and also the vias are to be recognized again over white circles. If you take a closer look at the diagram, you will notice that the via at $5V$ can be reached via the left or right MOSFET. However, the via at $0V$ can only be reached if both lower MOSFETs short-circuit. Thus, the structure is consistent with the circuit determined so far. The figure ## and figure 10 should show this in more details.

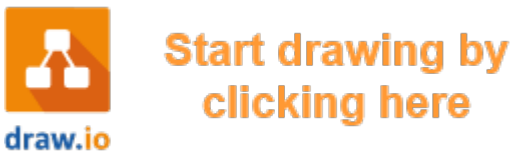


Fig. ##: NAND-Gatter in CMOS auf Chip

Fig. ##

Fig. 10: Simulation eines CMOS NAND-Gatters (Struktur ähnlich Si-Die)

eine einfache Rechnung

Fig. 12: Simulation eines Addierers

Wie können nun die NAND-Gatter so verschaltet werden, dass das Rechenwerk Operationen wie $y=a+b$ durchführen kann? Dazu wird die Operation zunächst nur für binäre Werte betrachtet. Um die binäre Größen von dezimalen Größen zu unterscheiden wird diesen ein $0b$ vorangestellt. Folgende Kombinationen sind also möglich:

- $0 + 0 = 0$ bzw. $0b0 + 0b0 = 0b00$
- $0 + 1 = 1$ bzw. $0b0 + 0b1 = 0b01$
- $1 + 0 = 1$ bzw. $0b1 + 0b0 = 0b01$
- $1 + 1 = 2$ bzw. $0b1 + 0b1 = 0b10$

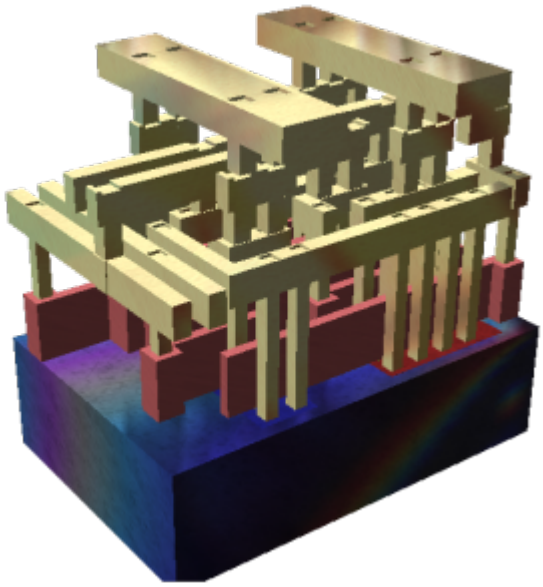
Es ist zu sehen, dass nur wenn beide Eingänge gerade 1 sind die zweite Stelle des Bitwertes gesetzt ist. Dies entspricht gerade einem AND. Da aber alles aus NAND-Gattern aufgebaut werden soll, muss eine geschickte Zusammenschaltung dieser Gatter gefunden werden. Hierzu wird einem NAND-Gatter ein Inverter-Gatter nachgeschalten. Das Inverter-Gatter wiederum erhält man über ein NAND-Gatter, wenn beide Eingänge verbunden werden. In figure 12 ist diese Schaltung unten durch die beiden unteren NAND-Gatter dargestellt.
Auch für die erste Stelle des Bitwertes lässt sich eine Schaltung finden. Wie kann man auf diese Schaltungen kommen? Dies wird im Kapitel Schaltnetze erklärt.

Werden viele Eingänge oder Ausgänge zusammengefasst, können größere Zahlenwerte umgesetzt werden. Das heißt, die Rechnung $3+3$ bzw. im binären $0b\color{green}1\color{violet}{1} + 0b\color{blue}{1}\color{red}{1}$ wird auf mehrere Einzelrechnungen heruntergebrochen. Dies ähnelt der händischen Addition durch Untereinanderschreiben der Zahlenwerte und schrittweiser Rechnung. In diesem Beispiel müsste zunächst $0b\color{violet}{1} + 0b\color{red}{1}$ berechnet werden, was $0b\boldsymbol{1}0$ ergibt. Im nächsten Schritt $0b\color{green}1 + 0b\color{blue}{1}$ muss zusätzlich noch der Überlauf aus der vorherigen Rechnung $0b\boldsymbol{1}$ berücksichtigt werden. So können prinzipiell beliebig lange Zahlen miteinander verknüpft werden.

Um die Zahlen handhabbarer für Mensch und Maschine zu gestalten wurde eine sinnvolle Gruppierung eingerichtet: 8 binäre Zahlenwerte werden zu einem Byte zusammengefasst. Dieses kann für Menschen lesbar als Dezimalwert $0...255$ oder Hexadezimalwert $0x00 ... 0xFF$ in Programmen dargestellt werden. Im Microprozessor werden diese Zahlenwerte stets als Binärwert gehandhabt.

Nano-Lego

Fig. ##



---

related Links

- Solver for Boolean functions: The solver specifies with which axioms Boolean equations can be simplified.
- Wolfram Alpha shows the different representations for boolean statements
- Silicon Zoo: Here you can see the practical implementation of logic gates in silicon.

- A nice overview of core ideas for calculating with electricity has been compiled by Gymnasium Kirchenfeld (CH, in German)
- http://www.righto.com/2016/12/die-photos-and-analysis-of_24.html
- Explanation of CMOS in the english Wikipedia
- Wiki page on integrated circuits

Applications

- Etherium: With this cryptocurrency, calculations on the blockchain are possible. Here, the basic logical functions are the most convenient - so a program should be feasible with as few boolean operators as possible
- Example in C for using of boolean algebra: by clicking on the Fork this button, the code can be changed.

Excercises

Details can be found in Introduction to Digital Systems

References

1. figure ##: Sbp@Wikimedia, CC BY-SA 4.0

2. figure 3: TravisGoodspeed@FlickrCC BY 2.0
3. figure 4: ZeptoBars@Wikimedia,CC BY 3.0
4. figure 2, figure ##: SiliconZoo.org, Lizenz unbekannt
5. figure ##: David Carron@Wikimedia,public domain

1)

to be correct: the voltage has to be between gate and source an, not at the gate terminal...